INTERNATIONAL STANDARD ISO/IEC 14496-2:2004
TECHNICAL CORRIGENDUM 4

Published 2010-09-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

Information technology — Coding of audio-visual objects —

Part 2: Visual

TECHNICAL CORRIGENDUM 4

Technologies de l'information — Codage des objets audiovisuels —

Partie 2: Codage visuel

RECTIFICATIF TECHNIQUE 4

Technical Corrigendum 4 to ISO/IEC 14496-2:2004 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Page 129, 6.2.13.9

Replace 6.2.13.9 with the following:

6.2.13.9 Studio Block

The detailed syntax for the term “DCT coefficient” is fully described in Clause 7.

StudioBlock(i) {	**No. of bits**	**Mnemonic**
if (pattern_code[i]) {		
if (macroblock_intra)		
if (i<4 || rgb_components == 1) {		
dct_dc_size_luminance	2-13	vlclbf

ICS 35.040.00 **Ref. No. ISO/IEC 14496-2:2004/Cor.4:2010(E)**

Published in Switzerland

if (dct_dc_size_luminance != 0)		
dct_dc_differential	1-18	vlclbf
if (dct_dc_size_luminance > 8)		
marker_bit	1	bslbf
} else {		
dct_dc_size_chrominance	2-13	vlclbf
if (dct_dc_size_chrominance != 0)		
dct_dc_differential	1-18	vlclbf
if (dct_dc_size_chrominance > 8)		
marker_bit	1	bslbf
}		
else		
First DCT coefficient	1-32	vlcfbf
while (next_bits() != End of block)		
Subsequent DCT coefficients	1-32	vlclbf
End of block	1-16	vlcfbf
}		
}		

Page 208, 6.3.13.2.9

Replace 6.3.13.2.9 with the following:

6.3.13.2.9 VLC code extension

In order to prevent start code emulation in StudioMacroblock() or StudioBlock(), a set of VLC codes that does not cause start code emulation for any combination of syntax elements shall be encoded.

load_vlc_code: This is a four bit integer indicating the presence of downloadable VLC codes for quantised AC coefficients. The value of load_intra_vlc_code and load_inter_vlc_code is derived from this code. Semantics are defined in the Table 6-80.

Table 6-80 — Meaning of load_vlc_code

load_vlc_code	Meaning
0000	Forbidden
0001	load_intra_vlc_code = 1 load_inter_vlc_code = 0
0010	load_intra_vlc_code = 0 load_inter_vlc_code = 1
0011	load_intra_vlc_code = 1 load_inter_vlc_code = 1
0100 – 1111	Reserved

If load_intra_vlc_code is set to '1', the VLC codes defined in Tables B.50 through B.61 shall be replaced with the codes indicated by intra_vlc_length[j][i] and intra_vlc_code[j][i]. If it is set to '0', there is no change in Tables B.50 to B.61.

If load_inter_vlc_code is set to '1', the VLC codes defined in Tables B.62 through B.73 shall be replaced with the codes indicated by inter_vlc_length[j][i] and inter_vlc_code[j][i]. If it is set to '0', there is no change in Tables B.62 to B.73.

intra_vlc_length[j][i]: This is a four bit unsigned integer indicating the length of the VLC code replacing the entry No. i in the Table No. j, with j=0 … 11 referring to the range of Tables B.50 through B.61. The actual length is intra_vlc_length[j][i]+1. The length of the VLC code of entry No. 21 in each table shall be limited to less than 14.

intra_vlc_code[j][i]: This is a 16 bit unsigned integer indicating the VLC code replacing the entry No. i in the Table No. j, with j=0 … 11 referring to the range of Tables B.50 through B.61. The most significant bits of the length indicated by intra_vlc_length[j][i] is the actual VLC code, and the remaining bits shall be filled with '1'.
inter_vlc_length[j][i]: This is a four bit unsigned integer indicating the length of the VLC code replacing the entry No. i in the Table No. j, with j=0 … 11 referring to the range of Tables B.62 through B.73. The actual length is inter_vlc_length[j][i]+1. The length of the VLC code of entry No. 21 in each table shall be limited to less than 14.
inter_vlc_code[j][i]: This is a 16 bit unsigned integer indicating the VLC code replacing the entry No. i in the Table No. j, with j=0 … 11 referring to the range of Tables B.62 through B.73. The most significant bits of the length indicated by intra_vlc_length[j][i] is the actual VLC code, and the remaining bits shall be filled with '1'.

If a table entry is not defined, then the VLC code extension shall contain a VLC code of specified VLC length for the undefined cases. This VLC code shall be parsed without influence on the decoding process These values shall not cause start code emulation for any combination of syntax elements. Table B.51 contains examples of undefined cases.

Page 289, 7.6.9.1

Replace the pseudo code with the following:

```
C[8] = { -8, 24, -48, 160, 160, -48, 24, -8 };

clip_ref(ref, x, y){
    return(ref[MIN(MAX(x, 0), component_width(ref) - 1)]
              [MIN(MAX(y, 0), component_height(ref) - 1)]);
}

mirror_ref(ref, width, height, x, y) {
    if (x < 0)
        x = -x - 1;
    if (y < 0)
        y = -y - 1;
    if (x >= width)
        x = width - (x - width) - 1;
    if (y >= height)
        y = height - (y - height) - 1;
    return(ref[x][y]);
}

clip_var( var) {
  if (var < 0)
    return 0;
  else if (var > 2^bits_per_pixel - 1)
    return 2^bits_per_pixel-1;
  else
    return var;
}

mc(pred,                    /* prediction block */
   ref,                     /* reference component */
   x, y,                    /* ref block coords for MV=(0, 0) */
   width, height,           /* prediction block dimensions */
   mvx, mvy,                /* half- or quarter-pel resolution motion vector */
   rounding,                /* rounding control (0 or 1 ) */
   pred_y0,                 /* field offset in pred blk (0 or 1) */
   ref_y0,                  /* field offset in ref blk (0 or 1) */
   y_incr,                  /* vertical increment (1 or 2) */
   color)                   /* color component ("y", "u" or "v") */
{
    if (color == "y")
```

```
        quarterPel = quarter_sample;
    else
        quarterPel = 0;

    if (quarterPel == 0) {
        dx = mvx >> 1;
        dy = y_incr * (mvy >> y_incr);
        if (mvy & y_incr)
            if (mvx & 1)
                for (iy = 0; iy < height; iy += y_incr)
                    for (ix = 0; ix < width; ix++) {
                        x_ref = x + dx + ix;
                        y_ref = y + dy + iy + ref_y0;
                        pred[ix][iy + pred_y0] =
                            (clip_ref(ref, x_ref + 0, y_ref + 0) +
                             clip_ref(ref, x_ref + 1, y_ref + 0) +
                             clip_ref(ref, x_ref + 0, y_ref + y_incr) +
                             clip_ref(ref, x_ref + 1, y_ref + y_incr) +
                             2 - rounding) >> 2;
                    }
            else
                for (iy = 0; iy < height; iy += y_incr)
                    for (ix = 0; ix < width; ix++) {
                        x_ref = x + dx + ix;
                        y_ref = y + dy + iy + ref_y0;
                        pred[ix][iy + pred_y0] =
                            (clip_ref(ref, x_ref, y_ref + 0) +
                             clip_ref(ref, x_ref, y_ref + y_incr) +
                             1 - rounding) >> 1;
                    }
        else
            if (mvx & 1)
                for (iy = 0; iy < height; iy += y_incr)
                    for (ix = 0; ix < width; ix++) {
                        x_ref = x + dx + ix;
                        y_ref = y + dy + iy + ref_y0;
                        pred[ix][iy + pred_y0] =
                            (clip_ref(ref, x_ref + 0, y_ref) +
                             clip_ref(ref, x_ref + 1, y_ref) +
                             1 - rounding) >> 1;
                    }
            else
                for (iy = 0; iy < height; iy += y_incr)
                    for (ix = 0; ix < width; ix++) {
                        x_ref = x + dx + ix;
                        y_ref = y + dy + iy + ref_y0;
                        pred[ix][iy + pred_y0] =
                            clip_ref(ref, x_ref, y_ref);
                    }
    } else {
        dx  = Div2Round(mvx) >> 1;
        dy  = y_incr * (Div2Round(mvy) >> y_incr);
        mvy = mvy >> (y_incr-1);

        for (iy = 0; iy < height + y_incr; iy += y_incr)
            for (ix = 0; ix < width +1; ix++) {
                x_ref = x + dx + ix;
                y_ref = y + dy + iy + ref_y0;
                PredMxN[ix][iy/y_incr] = clip_ref(ref, x_ref, y_ref);
            }
```

```
        if (mvx & 3) {
            for (iy = 0; iy < height/y_incr + 1; iy++)
                for (ix = 0; ix < width; ix++) {
                    sum = 0;
                    for (i = 0; i < 8; i++)
                        sum += C[i] *
                            mirror_ref(PredMxN, width+1, height/y_incr + 1, ix-3+i, iy);
                    TmpBlk[ix][iy] = clip_var((sum + 128 - rounding) >> 8);
                }
            if (mvx & 1)
                for (iy = 0; iy < height/y_incr + 1; iy++)
                    for (ix = 0; ix < width; ix++)
                        TmpBlk[ix][iy] = (TmpBlk[ix][iy] +
                                          PredMxN[ix+(mvx==3)][iy] +
                                          1 - rounding) >> 1;
            for (iy = 0; iy < height/y_incr + 1; iy ++)
                for (ix = 0; ix < width; ix++)
                    PredMxN[ix][iy] = TmpBlk[ix][iy];
        }

        if (mvy & 3) {
            for (ix = 0; ix < width; ix ++)
              for (iy = 0; iy < height/y_incr; iy++) {
                    sum = 0;
                    for (i = 0; i < 8; i++)
                        sum += C[i] *
                            mirror_ref(PredMxN, width, height/y_incr + 1, ix, iy-3+i);
                    TmpBlk[ix][iy] = clip_var((sum + 128 - rounding) >> 8);
                }
            if (mvy & 1)
                for (iy = 0; iy < height/y_incr + 1; iy++)
                    for (ix = 0; ix < width; ix++)
                        TmpBlk[ix][iy] = (TmpBlk[ix][iy] +
                                          PredMxN[ix][iy+(mvy==3)] +
                                          1 - rounding) >> 1;
            for (iy = 0; iy < height/y_incr; iy++)
                for (ix = 0; ix < width; ix++)
                    PredMxN[ix][iy] = TmpBlk[ix][iy];
        }

        for (iy = 0; iy < height; iy += y_incr)
            for (ix = 0; ix < width; ix++)
                pred[ix][iy + pred_y0] = PredMxN[ix][iy/y_incr];
    }
}
```

Page 290, 7.6.9.2

Replace the following pseudo code:

mc(Pf_Y, ref_Y_for, x, y, 16, 16, MVFx, MVFy, 0, 0, 0, 1);
mc(Pf_U, ref_U_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);
mc(Pf_V, ref_V_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);

with:

mc(Pf_Y, ref_Y_for, x, y, 16, 16, MVFx, MVFy, 0, 0, 0, 1, “y”);
mc(Pf_U, ref_U_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1, “u”);
mc(Pf_V, ref_V_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1, “v”);

Page 290, 7.6.9.3

Replace the following pseudo code:

```
mc(Pb_Y, ref_Y_back, x, y, 16, 16, MVBx, MVBy, 0, 0, 0, 1);
mc(Pb_U, ref_U_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);
mc(Pb_V, ref_V_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);
```

with:

```
mc(Pb_Y, ref_Y_back, x, y, 16, 16, MVBx, MVBy, 0, 0, 0, 1, “y”);
mc(Pb_U, ref_U_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1, “u”);
mc(Pb_V, ref_V_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1, “v”);
```

Page 290, 7.6.9.4

Replace the following pseudo code:

```
mc(Pf_Y, ref_Y_for, x, y, 16, 16, MVFx, MVFy, 0, 0, 0, 1);
mc(Pf_U, ref_U_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);
mc(Pf_V, ref_V_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);
mc(Pb_Y, ref_Y_back, x, y, 16, 16, MVBx, MVBy, 0, 0, 0, 1);
mc(Pb_U, ref_U_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);
mc(Pb_V, ref_V_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);
Pi_Y[i][j] = (Pf_Y[i][j] + Pb_Y[i][j] + 1)>>1;          i,j=0,1,2…15;
Pi_U[i][j] = (Pf_U[i][j] + Pb_U[i][j] + 1)>>1;          i,j=0,1,2…7;
Pi_V[i][j] = (Pf_V[i][j] + Pb_V[i][j] + 1)>>1;          i,j=0,1,2…7;
```

with:

```
mc(Pf_Y, ref_Y_for, x, y, 16, 16, MVFx, MVFy, 0, 0, 0, 1, “y”);
mc(Pf_U, ref_U_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1, “u”);
mc(Pf_V, ref_V_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1, “v”);
mc(Pb_Y, ref_Y_back, x, y, 16, 16, MVBx, MVBy, 0, 0, 0, 1, “y”);
mc(Pb_U, ref_U_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1, “u”);
mc(Pb_V, ref_V_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1, “v”);
Pi_Y[i][j] = (Pf_Y[i][j] + Pb_Y[i][j] + 1)>>1;          i,j=0,1,2…15;
Pi_U[i][j] = (Pf_U[i][j] + Pb_U[i][j] + 1)>>1;          i,j=0,1,2…7;
Pi_V[i][j] = (Pf_V[i][j] + Pb_V[i][j] + 1)>>1;          i,j=0,1,2…7;
```

Page 301, 7.7.2.1

Replace the following pseudo code:

```
field_motion_compensate_one_reference(
    luma_pred, cb_pred, cr_pred, /* Prediction component pel array */
    luma_ref, cb_ref, cr_ref,    /* Reference VOP pel arrays */
    mv_top_x, mv_top_y,           /* top field motion vector */
    mv_bot_x, mv_bot_y,          /* bottom field motion vector */
    top_field_ref,               /* top field reference */
    bottom_field_ref,            /* bottom field reference */
    x, y,                        /* current luma macroblock coords */
```

```
    rounding_type)                /* rounding type */
{
    mc(luma_pred, luma_ref, x, y, 16, 16, mv_top_x, mv_top_y,
        rounding_type, 0, top_field_ref, 2);
    mc(luma_pred, luma_ref, x, y, 16, 16, mv_bot_x, mv_bot_y,
        rounding_type, 1, bottom_field_ref, 2);
    mc(cb_pred, cb_ref, x/2, y/2, 8, 8,
        Div2Round(mv_top_x), Div2Round(mv_top_y),
        rounding_type, 0, top_field_ref, 2);
    mc(cr_pred, cr_ref, x/2, y/2, 8, 8,
        Div2Round(mv_top_x), Div2Round(mv_top_y),
        rounding_type, 0, top_field_ref, 2);
    mc(cb_pred, cb_ref, x/2, y/2, 8, 8,
        Div2Round(mv_bot_x), Div2Round(mv_bot_y),
        rounding_type, 1, bottom_field_ref, 2);
    mc(cr_pred, cr_ref, x/2, y/2, 8, 8,
        Div2Round(mv_bot_x), Div2Round(mv_bot_y),
        rounding_type, 1, bottom_field_ref, 2);
}
```

with:

```
field_motion_compensate_one_reference(
    luma_pred, cb_pred, cr_pred, /* Prediction component pel array */
    luma_ref, cb_ref, cr_ref,    /* Reference VOP pel arrays */
    mv_top_x, mv_top_y,           /* top field motion vector */
    mv_bot_x, mv_bot_y,           /* bottom field motion vector */
    top_field_ref,                /* top field reference */
    bottom_field_ref,             /* bottom field reference */
    x, y,                         /* current luma macroblock coords */
    rounding_type)               /* rounding type */
{
    mc(luma_pred, luma_ref, x, y, 16, 16, mv_top_x, mv_top_y,
        rounding_type, 0, top_field_ref, 2, "y");
    mc(luma_pred, luma_ref, x, y, 16, 16, mv_bot_x, mv_bot_y,
        rounding_type, 1, bottom_field_ref, 2, "y");
    mc(cb_pred, cb_ref, x/2, y/2, 8, 8,
        Div2Round(mv_top_x), Div2Round(mv_top_y/2)*2,
        rounding_type, 0, top_field_ref, 2, "u");
    mc(cr_pred, cr_ref, x/2, y/2, 8, 8,
        Div2Round(mv_top_x), Div2Round(mv_top_y/2)*2,
        rounding_type, 0, top_field_ref, 2, "u");
    mc(cb_pred, cb_ref, x/2, y/2, 8, 8,
        Div2Round(mv_bot_x), Div2Round(mv_bot_y/2)*2,
        rounding_type, 1, bottom_field_ref, 2, "v");
    mc(cr_pred, cr_ref, x/2, y/2, 8, 8,
        Div2Round(mv_bot_x), Div2Round(mv_bot_y/2)*2,
        rounding_type, 1, bottom_field_ref, 2, "v");
}
```

Page 302, 7.7.2.2

Replace the following pseudo code:

```
    PMV[0].x = PMV[0].x + MVD[0].x;
    PMV[0].y = 2 * (PMV[0].y / 2 + MVD[0].y);
    PMV[1].x = PMV[1].x + MVD[1].x;
    PMV[1].y = 2 * (PMV[1].y / 2 + MVD[1].y);
    field_motion_compensate_one_reference(
        luma_pred, cb_pred, cr_pred,
        luma_fwd_ref_vop, cb_fwd_ref_vop, cr_fwd_ref_vop,
        PMV[0].x, PMV[0].y, PMV[1].x, PMV[1].y,
        forward_top_field_reference,
        forward_bottom_field_reference,
        x, y, 0);
```

with:

```
    r_size = vop_fcode - 1
    f = 1 << r_size
    high = ( 32 * f ) -  1;
    low = ( (-32) * f );
    range = ( 64 * f );
    PMV[0].x = PMV[0].x + MVD[0].x;
    PMV[0].y = 2 * (PMV[0].y / 2 + MVD[0].y);
    PMV[1].x = PMV[1].x + MVD[1].x;
    PMV[1].y = 2 * (PMV[1].y / 2 + MVD[1].y);
    for (mv = 0; mv < 2; mv++) {
        if ( PMV[mv].x < low )
            PMV[mv].x = PMV[mv].x + range;
        if (PMV[mv].x > high)
            PMV[mv].x = PMV[mv].x - range;
        if (PMV[mv].y < low )
            PMV[mv].y = PMV[mv].y + range;
        if (PMV[mv].y > high)
            PMV[mv].y = PMV[mv].y - range;
    }
    field_motion_compensate_one_reference(
        luma_pred, cb_pred, cr_pred,
        luma_fwd_ref_vop, cb_fwd_ref_vop, cr_fwd_ref_vop,
        PMV[0].x, PMV[0].y, PMV[1].x, PMV[1].y,
        forward_top_field_reference,
        forward_bottom_field_reference,
        x, y, 0);
```

Page 303, 7.7.2.2

Replace the following pseudo code:

```
    PMV[2].x = PMV[2].x + MVD[0].x;
    PMV[2].y = 2 * (PMV[2].y / 2 + MVD[0].y);
    PMV[3].x = PMV[1].x + MVD[1].x;
    PMV[3].y = 2 * (PMV[3].y / 2 + MVD[1].y);
    field_motion_compensate_one_reference(
        luma_pred, cb_pred, cr_pred,
```